G-WINDOWS

G-WINMGR-0490.01

Table Of Contents

Part A: USING G-WINDOWS

The full power of a multi-tasking system is usually hidden to a single user developing software under OS-9/68K because all interaction must take place through a simple ASCII terminal. Normally, only one process at a time may communicate with the user through a text terminal.

In an attempt to allow more access to the computer, some terminal managers have been written which allow a user to access several processes at once by supporting 'paging'. 'Paging' tricks the computer into thinking that several terminals are attached to it and lets the user view only one of these virtual terminals at a time. These virtual terminals are a big improvement over a single terminal, but they still only allow one process to be viewed at a time.

This window manager was designed to allow multiple tasks to be viewed and accessed at the same time under OS-9/68K. It is loosely based on work done by XEROX at its Palo Alto research center several eons ago, as are the windowing systems of all small computers. Specifically, many of its features were borrowed from the capabilities of X-Windows.

In addition, the window manager allows you to easily add pop-up menus, alert messages, and request boxes to your applications.

The window manager works with a graphics screen as its base, and allows the user to partition the screen into several 'windows', through which each process can be accessed. The windows may contain either text, graphics, or a combination of the two.

Each process may treat its window as if it were an entire text and graphics terminal without worrying about interference from other windows. A process may open more than one window and access several virtual graphics terminals at once.

Windows may be different sizes, may overlap, and may be converted to a small icon to reduce screen clutter. Windows may be moved and their size changed by moving a pointer on the screen in conjunction with a mouse.

All processes attached to windows may output text or graphics simultaneously, even if their windows are partially obscured. Keyboard and mouse input is routed to any of the processes with a click of the mouse button.

Chapter 1: Starting the Window Manager

Before starting the window manager, the window manager files 'wfm', 'qfonts' and the OS-9/68K files 'scf', 'nil', and 'null' should be loaded into memory. Additional modules that must be loaded are the window descriptor (usually 'win'), the crt driver and descriptor, the keyboard driver and descriptor, and the mouse driver and descriptor.

The window manager is started automatically when the first window is created.

Chapter 2: Using the Window Manager

Windows

Opening a Window

Windows are created simply by opening a path to the device descriptor (usually '/win').

Windows may be created from a shell command line by using the output redirection facilities of the shell. For instance, a directory listing may be sent to a window by using the command: "dir >/win".

A new window is created each time a path to '/win' is opened, even from within the same process. In this respect windows are handled like OS-9/68K pipes. Windows may be shared between processes by inheriting the I/O path from a common parent. Another way of allowing several processes to access the same window is by creating a named window.

Named Windows

Named windows may be created in the same way that named pipes may be created under OS-9/68K. A named window is created by opening a path to '/win/<name>', where <name> is a name which adheres to the standard OS-9/68K naming conventions.

Opening a named window which doesn't exist will create it. Trying to create a named window which already exists will generate a 218 error (creating an existing file). For instance, the following will generate an error on the second directory:

```
dir >/win/fred
dir >/win/fred
```

The following demonstrates the proper way to send output to a named window which already exists:

```
dir >/win/fred
dir >+/win/fred
```

A list of named windows may be obtained by getting a directory of '/win': "`dir -e /win`". Shell pattern matching may even be used with named windows: "`echo -n /win/f*`".

There are two types of windows available to application programs: standard windows and custom windows. Other than the way the windows are set up initially, there is only one real difference between them.

Standard Windows

Standard windows are generated by creating them with read or write permissions only. For example: `open("win",3)`, `creat("/win",3)`, or `create("/win",3,3)`. This is the type of window the shell creates with I/O redirection.

Standard windows are automatically set up for terminal emulation with escape sequences as defined later in this document. All text written to the window using normal OS-9/68K calls is remembered by the window manager so the application needs to do no window refreshing.

Standard windows initially have the text cursor turned on and have all the standard window gadgets attached.

Standard windows are enabled (made visible) as soon as a read or write to the window takes place.

If text has been written to a standard window, and no reads or special window function calls have been made, the window will not disappear when the application closes the path to it. When a window no longer has a process attached to it, the title in the drag bar will disappear. You can get rid of a dead window by pressing the close button. This is the big difference between standard and custom windows.

The WINDOW Environment Variable

When a standard window is created, several of its attributes may be defined through the use of the WINDOW environment variable.

Each attribute in the environment variable is assigned a numerical value, and assignments are separated from each other with a comma. For example, to set the text size of the next window created to 100 columns by 32 rows, enter the following on a shell command line:

```
Setenv WINDOW TX=100,TY=32
```

where TX stands for window width in characters and TY stands for window height in characters.

Values may be entered in hexadecimal format by preceding the number with "$", "0x", or "x".

The following are the attributes which may be set in the WINDOW environment variable:

- BS Keyboard code to use as the backspace key. (default = $7f)
- CD Keyboard code to use as the down arrow key. (default = $0a)
- CL Keyboard code to use as the left arrow key. (default = $08)
- CR Keyboard code to use as the right arrow key. (default = $0c)
- CU Keyboard code to use as the up arrow key. (default = $0b)
- FR Default frame type for standard windows
  (0=no frame, 1=bevelled edge, 2=rounded edge).
- LB Keyboard code which simulates left mouse button. (default = $00)
- PX X position of top left corner of window. (default = random)
- PY Y position of top left corner of window. (default = random)
- QF Quick Font to use for terminal emulation.
  -1 = normal quick font (default)
  -2 = small quick font
  -3 = large quick font
- SX Outside window width in pixels. (default = 1/2 CRT width)
- SY Outside window height in pixels. (default = 1/2 CRT height)
- TB RGB value to use as text background color.
- TF RGB value to use as text foreground color.
- TX Character width of user area. (default = 80)
- TY Character height of user area. (default = 25)
- UK Flag to set G0 character set to UK ANSI.

The RGB values specified for 'TB' and 'TF' must be in the hexadecimal form $00BBGGRR where $000000 is black, $ffffff is white, and $2098ff is amber. On fixed color systems, the window manager will use the color which most closely matches the color specified.

The 'LB' environment variable defines a key which simulates the mouse left button down/up sequence. By default, it is set to $00 (Ctl-@). If your keyboard supports it, I recommend using $A0 (Alt-<space>).

LEFT and RIGHT Cursor Keys

During `readln()`, the LEFT and RIGHT cursor keys of most keyboards may be used to move the text cursor within the line being entered. This allows characters to be added and deleted from the middle of the line rather than just the end, as SCF does. On those keyboards which use the BACKSPACE key (ASCII $08) as the LEFT arrow key, it may not be used to delete the character left of the text cursor. For this reason, the DELETE key (ASCII $7f) is used as the default backspace key.

Readln() History

The last fifty text lines entered through the OS-9/68K `readln()` call are remembered by the window manager and may be recalled and edited by using the UP and DOWN arrow keys on most keyboards. Pressing the UP arrow key will cause the current line to be replaced by the line entered just before it. Pressing the DOWN arrow key will cause the currently displayed line to be replaced by the line entered just after it.

The history is limited to lines containing three or more characters.

Expansion of File Names

During a `readln()`, if a partial file name is entered, pressing <ESC> will complete the file name for you. If more than one matching file exists, the name is filled out as much as possible, and pressing <ESC> a second time displays a list of all matching files.

Custom Windows

Custom windows are created by creating them with the execute bit set. For example: `create("/win",3,7).`

Initially, custom windows do not remember text written to them, so the application writing to them has the responsibility of refreshing them when the window manager requests it.

Custom windows initially have the text cursor turned off and have no window gadgets or even a border around the window.

Custom windows are disabled (not visible) until the application enables them.

Custom windows always disappear when all paths to them are closed.

Window Parts

In its simplest form, a window is just a rectangular area on the screen which may be modified by the application which created it. Most of the time, windows are created with a 16 pixel frame and several 'gadgets' attached to it. The gadgets allow a user to control several aspects of the way the window is displayed and help make your application program much friendlier.

Window User Area

The user area of the window is that part of the window that an application program can modify. It covers the entire window except for the 16 pixel frame and the window gadgets (if they are present).

Pixels are addressed in the user area using the Cartesian coordinate system. Visible X positions start at the left of the user area with x=0 and increase in value further to the right. Visible Y positions start at the bottom of the user area with y=0 and increase in value further to the top.

The user area may be any size up to 32767 wide and 32767 tall. The portion of the user area visible within the window might be only a small portion of the entire area.

If no scroll bars are attached to the window, the size of the user area may be set to follow the size of the window frame. When this is done, the origin of the user area is always placed at the first visible pixel in the lower left corner of the window. The size of the user area is limited to what is visible within the window borders. This method is used by the 'clock' demo program to allow the clock face to be stretched to any size.

Mouse Buttons In the User Area

In the user area of custom windows, the effects of clicking the mouse buttons are defined by the program which created them. In standard windows, however, the left mouse button is used to control copying and pasting text between windows.

To copy a section of text from the window to the text copy buffer, use the mouse to point to the start of the text to be copied, then press the left button and hold it while moving the mouse pointer to the end of the text. The selected text will appear inverted. When the button is released, the selected text is copied to the text copy buffer. The text will remain highlighted until the window is written to.

To retrieve text from the text copy buffer, simply double click on a standard window. The text will be presented to the program attached to the window as if it were entered through the keyboard. Text copied from one window may be pasted to another, even if the two programs have nothing to do with each other.

It is important to remember that not all programs write to all areas of the screen, and may not write TABs to the screen when you expect them to. Text editors are prime examples of this. Since the text copy and paste can only work with the information actually written to the window, you don't always get the expected results.

An extra feature is added when uMacs is being used in a window. Pointing at a letter and single clicking the left mouse button will cause the text cursor to be moved to that spot. This is done by sending uMacs cursor move commands (Ctl-N, Ctl-P, etc.) as though they were typed on the keyboard.

There are a some things to watch out for when using this feature with uMacs. Since uMacs does not write TABs or all spaces to the screen, clicking on a space may cause the cursor to move to the wrong spot. If a long line is being edited past the right of the screen, clicking on that line to move the cursor will cause it to go to the wrong spot.

A helpful hint: to paste the window text buffer in uMacs without repositioning the text cursor when you click the mouse, double click the mouse on or below the uMacs status line.

Window Gadget Area

If any of the following gadgets are attached to a window, a 16 pixel frame is automatically added to the window.

Close Button

The close button is placed in the upper left corner of the window. It looks like an archery target (for no good reason).

Its function is determined by the application program, but normally it is used to terminate the program and close the window. The window does not have to be the active window for the close button to work.

If it is attached to a standard window, its function changes when a read takes place. Before a read takes place, clicking the close button will cause the window manager to kill the last process to write to the window (provided that process still has a path to the window open). After a read takes place, clicking the close button will cause the window manager to return an EOF error to any program currently reading from the window.

If uMacs is running in a standard window, it is treated as a special case and ESC-z is sent to it instead of EOF.

If all paths have been closed to the window and it still hangs around (see the description of standard windows), clicking the close button will remove the window.

Drag Bar

The drag bar extends the length of the top of the window, except for the space taken by other buttons. The drag bar serves several purposes.

The bars in the drag bar will be drawn as solid when the window is the current active window. Only the current active window may receive keyboard or mouse input. If the window is not active, clicking on the drag bar will bring it to the front of the window stack and activate it.

The window title is centered in the drag bar. Normally, the title consists of the process id and module name of the last program to access the window. If it is a named window, the window name is included in the title as well. An application may override the default title with its own.

If the window is the current active window, pressing and holding the left mouse button on the drag bar will allow you to move the window to a new location. Release the button when the dashed frame is in the wanted location.

Hibernate Button

The hibernate button is located just to the right of the drag bar and can be identified by the big 'Z'. Clicking the left mouse button on the hibernate button will cause the window to disappear, and in its place an icon will appear in the lower right corner of the screen. The window can be restored by clicking the left mouse button on the icon.

The window need not be the active window to use the hibernate button.

'Hibernating' a window is used to reduce screen clutter when you aren't currently interested in a window's contents. For instance, you might have a couple of shells, a uMacs, and a compilation going on in separate windows. Instead of having all of them compete for screen space, a shell and the compilation could be 'hibernated' to get them out of the way.

A program is not stopped when its window is hibernating. It can still print text and graphics to the window. If is waiting for input, the default hibernate icon will change from a 'thinking balloon' to a 'talking balloon'.

An application may customize its hibernate icon through the `'Window_Set()'` function described in the G-WINDOWS developer's manual.

If you do not have access to the source code to a program, you can still provide a custom hibernate icon (this is described later in this document).

Get Big Button

The 'get big' button is located to the right of the hibernate button and to the left of the back button. It is represented by four arrows pointing outward. The get big button only works on the currently active window, otherwise it causes the window to become the currently active window.

Clicking the left mouse button on the get big button causes the window to be expanded to its largest allowed size. At that time the 'get big' icon changes to the 'get small' icon, which is represented by four arrows pointing inward. Clicking the 'get small' button will return the window to its previous size.

Back Button

The back button is located in the upper right corner of the window. Clicking the left mouse button on the back button will move the window to the bottom of the window stack (see the window stack section). This may cause the window to be obscured by other windows. If the window was the current active window, it is deactivated and the window at the top of the stack is activated.

The window does not have to be the active window for the back button to be used.

Size Button

The size button is located at the bottom right corner of the window. Clicking and holding the left mouse button on the size button allows the user to change the size of the window.

If the window is not the active window, then clicking on the size button brings the window to the top of the window stack and activates it.

Scroll Bars

There can be two scroll bars attached to a window, a vertical scroll bar on the right side of the window, and a horizontal scroll bar on the bottom of the window. Scroll bars allow the user to view different parts of the window user area when the window is too small to display it in its entirety.

If the window is not the active window, then clicking on a scroll bar brings the window to the top of the window stack and activates it.

Each scroll bar is made up of 5 parts: two scroll arrows, two page boxes, and a positioning bar.

At each end of the scroll bar is an arrow box. Clicking the left mouse button on an arrow box will scroll the window to let you view more of the user area in that direction. The number of pixels scrolled is usually set to the height of the default font, but an application program may set the scroll amount for its own purposes.

The space between the scroll arrows is divided between the page boxes and the positioning bar.

The positioning bar (the raised box) varies in size depending on how much of the user area is visible in the window. If the entire user area is visible, the positioning bar fills the entire area and may not be moved. Otherwise, the positioning bar shows which part of the user area is visible, and the paging boxes (the depressed portion) show which parts of the user area are not visible.

By pressing and holding the left mouse button on the positioning bar, you can select which portion of the user area that will be visible in the window.

Clicking on the paging areas will scroll the view of the user area by the number of pixels currently visible in the window.

Window Stack

Because windows may be placed anywhere on the screen and they may overlap, some windows will be 'in front' of other windows. This ordering is referred to as the window stack. Windows are said to be higher in the stack or in front of other windows when they obscure other windows. Windows are said to be lower in the stack or behind other windows when they are partially hidden by those windows.

Stack ordering takes place even when no windows are overlapping each other.

The window with the topmost position in the stack is called the active window. The active window is the window to which all keyboard and mouse input is routed. The drag bar of the active window is drawn with solid black bars.

A window's position in the window stack may be adjusted by clicking the left mouse button on the back button to send it to the bottom of the stack, or by clicking on the user area (and some gadgets) to bring it to the top of the stack.

Window Priorities

There are some limitations to the position of a window in the window stack. Each window has associated with it a stack priority. This can range in value from 0 through 7. A window's stack priority is initially set to 3 and may be changed with the '`Window_Set()`' function.

A description of each of the stack priorities follow. This is presented in the 'using windows' section because it may affect the way windows act.

Level 0: Forces a window to always be shown behind all other windows. If a window has a priority of 0, it may be the active input window and still be

obscured. Only one window may have a priority of 0. This is used by the desktop manager to create the window which contains the disk icons.

Level 1: This is another special priority. Only the window manager can set a window to priority 1. Hibernating windows are automatically set to level 1 and then back when they are awakened. Having a priority of 1 causes a window to be behind all windows except the backdrop (priority 0) window.

Levels 2-6: These priorities affect only those windows owned by the same program. Among windows owned by the same program, those with the highest priority are the only ones which may be at the top of the stack and active for input. Windows with these priorities not owned by the same process are treated as if they have the same priority.

Level 6 is used by the simple alerts and gadget alerts when creating their message windows if the message is important to the application but not to the system in general.

Level 7: This is another special priority. Windows with this priority are always at the top of the window stack regardless of which process owns them. If there is more than one window with a priority of 7, each one may be made the active window.

Level 7 is used by alerts when creating their message windows if the message is important to the entire system and must be responded to immediately.

Menus

Application programs with a path open to a window can make use of the built-in menuing capabilities of the window manager to simplify their user interface.

If a program uses menus, its menu can be activated whenever its window is the active input window. Clicking the right mouse button causes the menu to appear at the same location as the mouse pointer. A list of menu choices appear, some with arrows by them and some with bracketed letters beside them.

Some programs which use menus may have different menus attached to different parts of the window. If the mouse pointer is inside one of these 'hot spots' or 'transparent windows', a different menu may appear when the right mouse button is pressed than on other parts of the window.

As the mouse pointer is moved over the choices, the active choices will be inverted. To select a menu choice, click the left or right mouse button when a choice is inverted. To select no menu choice, click the left or right mouse button when no choice is inverted.

If a menu item has an arrow next to it, moving the pointer onto the arrow will cause a child menu to appear.

While the menu is displayed on the screen, all other output to the screen is stopped.

Faded Choices

Some menu choices will appear faded and hard to read. These are choices that the application program has decided to make inactive for the moment. They are kept in the menu structure as place holders so the menu

does not change drastically while using it. These faded choices will not be shown as inverted when the mouse pointer passes over them and they are not selectable menu choices.

Check Marks

Some menu choices may have check marks next to them. These choices act like switches to turn a feature of the application program on or off. When the check mark is present, the named feature is turned on, otherwise it is turned off. Select the menu item to toggle the check mark on and off.

Walking Menus

Some menu choices may have arrows next to them. These are not actually menu choices, but an entire category of menu choices. By moving the mouse pointer onto the arrow, a sub-menu will appear which contains a list of related menu choices. This menu acts the same as the original menu except that it disappears when the mouse pointer moves onto its parent menu.

Use of walking menus can greatly simplify the use of a menu with a lot of choices.

Keyboard Activation of Menu

Some menu choices may have a letter in angle brackets next to them. These menu choices may be selected without even making the menu appear on the screen. Just hold the <Ctrl> key and press the letter in the brackets, and the choice will be selected just as if you had used the mouse to make the selection.

This feature makes programs easier and quicker to use once you are familiar with the menu commands.

Alerts

Alerts are simply windows that pop-up while an application is running, which ask for some kind of response from the user. The response must be supplied before the application program can continue. Alerts can be messages from the application or requests for information.

There are two kinds of alerts used with G-WINDOWS: simple alerts that do not use G-WINDOW gadgets, and more powerful gadget alerts that make use of G-WINDOW gadgets.

Simple Alerts

Simple alerts do not use G-WINDOW gadgets and are less powerful as a result. They are available for use in the rare case that a developer does not want to use gadget modules.

Simple alerts serve three different needs: a message to be acknowledged, a quick yes/no type of response, or a single line of text input.

For message and yes/no alerts, the user responds buy clicking the mouse on one of the buttons at the bottom of the alert. Pressing <RETURN> automatically chooses the rightmost button.

When a program wants a single line of text, a text entry box will also be included in the alert.

A cursor (a simple vertical line) shows where the next character you type will be inserted. The cursor may be repositioned with keyboard arrow keys or by pointing at where you want it to go and clicking the left mouse button.

`Ctl-X` will delete the whole line. `<Delete>` will delete the character to the left of the cursor. Pressing return or clicking one of the boxes at the bottom of the alert completes the data entry.

Gadget Based Alerts

Gadget based alerts are more powerful than simple alerts, but require G-WINDOW gadget modules to be available. See the section describing G-WINDOW gadgets for more information concerning gadgets.

Standard Gadget Alerts

Standard gadget alerts perform the same functions as simple alerts, except that up to six buttons and eight text input fields may included in every gadget.

In addition to the text editing features supported by simple alerts, each gadget text input field also supports several extra features that make text entry easier.

`Ctl-A` and `Ctl-E` move the text cursor to the beginning and end of the text line. `Ctl-D` deletes the character to the right of the cursor.

By pressing and holding the left mouse button while dragging the mouse, text may be marked. When the mouse button is released, the text is copied into the global text copy buffer used by the window manager. The copied text may then be entered into any window or text field by double clicking the mouse in the destination window.

When text is marked for copy into the global text copy buffer, an extra feature comes into play. Any key press that would change the text will delete the marked area and moves the text cursor to the point that the marked text occupied. This eliminates a lot of backspacing when you don't want to use `Ctl-X`.

File Browser Gadget Alert

When an application needs the name of a file, it may use the G-WINDOWS file browser.

On the bottom of the left side of the browser are the OK and CANCEL buttons. Pressing OK or the <RETURN> key will pass the chosen directory and filename back to the application program.

On the top of the left side of the browser are three fields which contain the directory, filename mask, and filename.

The right side of the file browser displays a list of files contained in the current directory. If needed the list can be scrolled up and down to display all the files. Directory name are followed by '/' to differentiate them from ordinary files. Only files which the application is interested in will appear in the list.

Clicking the mouse button once in the file list will copy the name of the file that was clicked on into the file name edit field. Double clicking on a file name will also automatically press the OK button as well.

The filename field is an editable text field. File or directory names may be entered here. If a directory name is entered and the OK button pressed, the file browser will switch to that directory and display the files it contains. If a partial file name is entered, pressing the <ESC> key will automatically expand the file name to match what is present on disk.

The filename mask is an editable text field. It is used to limit the number of files which are displayed in the file list at the right of the browser. It uses the characters '?' and '*' in the same way as OS-9/68K shell pattern matching. Only those files which match the specified pattern will appear in the file name list on the right side of the browser.

The directory field may not be directly edited. It displays the last section of the entire directory pathlist. Pressing and holding the left mouse button on the directory field will display all segments of the directory pathlist.

By moving the cursor down the pathlist and releasing the button on one of the segments, one can quickly switch the browser into a directory closer to the root device. Using this method, the browser can even display a module directory of the system so a different disk device can be selected.

The file browser can be used to select files on another OS-9/68K NFM network node. Simply change directories to the module level, double click on the desired network node, and then double click on the desired disk device name.

Since OS-9/68K does not provide a method of determining what devices are available on another network node, the following devices are always assumed to exist: '/d0', '/d1', '/dd', '/h0', and '/r0'. If another device is available, it may be selected by entering its name into the file name entry field and pressing <RETURN>.

Chapter 3: Custom Hibernate Icons

The best way to customize a hibernate icon for a window is with the 'Window_Set()' function call directly from the application. Unfortunately, most programs written for OS-9/68K are not yet equipped to take advantage of the window manager. Even without access to the source code for a program, a custom hibernate icon can be attached to it through a simple binary icon file.

All binary files are looked for in the directory '.ICONS' located in the root directory of the desk device specified in the OS-9/68K init module on your system. If your default device is '/h0', the window manager will look in the directory '/h0/.ICONS' for the icon files. If it doesn't find the wanted file there, it then looks in the directory '/dd/.ICONS'. If the window manager still can't match an icon file to the program attached to the window, it uses the default hibernate icon (thinking and talking balloons).

The icon file must have the same name as the program module that has the window open. For example, the name of the file icon intended for the OS-9/68K shell would be '/h0/.ICONS/shell'.

The first four bytes are a long integer which contains the following flags:

Bit 0	Include the window title in the icon.
Bit 1	A 'waiting for input' icon is included along with the 'busy' icon.
Bit 2	Right justify window title within the icon.
Bits 3-31	must be set to zero.

The default or 'busy' icon follows the flags integer. The 64 by 64 pixel icon is stored as 4 columns of 64 words each. If a 'waiting for input' icon is supplied, it follows immediately after the 'busy' icon. The size of the file should be either 516 or 1028 bytes.

Source code for an example hibernate icon file is provided in appendix B.

Chapter 4: Colors

The window manager will look different and have different capabilities depending on the number and type of colors which are supported by the graphics hardware being used. For instance, systems with a color look-up table will be much more flexible than systems with a TTL or monochrome display.

The window manager is designed to work with raster based graphics systems which have up to 8 bits per pixel (256 colors at once on the CRT). So far, it has been used only on systems with 1, 4, and 8 bits per pixel (2, 16, and 256 colors).

The different types of displays and the different capabilities and limitations imposed on them are as follows:

Monochrome 1 bit per pixel
- The 'dim' and 'blink' VT100 text attributes are not available.
- The window frame is flat with bevelled edges.
- Raster images with depths of 4 and 8 bits per pixel may not be displayed.
- Multi-color fonts may not be displayed.

16 color TTL 4 bits per pixel
- The 'blink' VT100 text attribute is not available.
- The window frame is flat with bevelled edges.
- Raster images with depths of 8 bits per pixel may not be displayed.
- 8 bit multi-color fonts may not be displayed.

16 colors with CLUT 4 bits per pixel
- The window manager reserves 5 colors for text, frame, and icon display. These colors may be used by programs, but not changed.
- Raster images with depths of 8 bits per pixel may not be displayed.
- 8 bit multi-color fonts may not be displayed.

256 colors with CLUT 8 bits per pixel

The window manager reserves 8 colors for text, frame, and icon display. These colors may be used by programs, but not changed.

When a color look-up table is available, portions of the CLUT must be allocated to each window that uses it. It is unreasonable to expect all windows to share the same CLUT, or for a single window to prevent other windows from getting the number of colorsthey need.

The window manager avoids these limitations by allowing windows to share sections of the CLUT. If a section of CLUT is shared by multiple windows, only the window which is active for input is assured to be displayed with the correct colors. This is most often visible on 16 color CLUT systems.

When the colors in an inactive window look messed up, the CLUT sharing is usually the reason. Remember: this is a feature, not a bug. CLUT sharing may be avoided by having a program specify the exact portion of the CLUT to be used.

Chapter 5: Font Modules

G-WINDOWS supports multiple fonts in different sizes.

G-WINDOW fonts are stored in OS-9/68K data modules. Font modules are automatically shared between multiple processes to save memory. Like any G-WINDOW modules, they may be placed in ROM.

There are two type of font modules used by G-WINDOWS: 'general' fonts and 'quick' fonts. G-WINDOW quick fonts are stored in a module named 'qfonts', which must be present in memory for G-WINDOWS to run. The 'qfonts' module holds three fonts which have the following limitations:

1) maximum size of 8 wide by 16 tall, including horizontal spacing.
2) mono spaced only.
3) monochrome only.
4) supports codes $00 through $7f only.

The 'qfonts' fonts are used by the window manager for terminal emulation, window titles, menus, and icon text. It is also used by the desktop manager for all of its text output. They are always present and may be used by application programs. The first quick font is normally 6 pixels wide by 13 tall. The second is smaller, normally 5 wide by 10 tall. The last quick font is the largest, and is normally 8 pixels wide by 16 tall. Source to the 'qfonts' module is supplied so it may be modified to meet special needs.

The remainder of this section deals only with general fonts.

G-WINDOW general fonts, like almost all fonts used by other windowing systems, are built with pixels rather than curves. Because of this, each size of a given font requires a whole new set of data to define it. This is why each font has a limited number of sizes available. This limitation is acceptable due to the higher speed and quality of the fonts.

The size of a font is defined by its point size. The point size of the font is measured from the lowest pixel contained in the font to the highest pixel contained in the font, except for diacritical marks like the tilde or umlaut.

Any number of fonts and font sizes may be placed in a single font module. Normally, however, only one font in a single size is placed in each module. Since there is little overhead when storing a single font in each module, this saves a lot of memory in general use.

Font Storage on Disk

In order to be easily recognized and loaded by programs running under G-WINDOWS, the following method has been defined for storing fonts on disk. It is not required, but will greatly ease the use of fonts with G-WINDOWS if adhered to.

If a disk device is available, font modules are stored in the directory 'FONTS' on the disk named in the OS-9/68K init module. The directory '/dd/FONTS' is used if no disk is specified in the init module. Each font has its own sub-directory, which itself has the same name as the font stored in it. Within the font sub-directory, each size of the font is stored in its own font module file. Both the file and the font module contained within it are given the name created by appending an underscore and the point size of the font to the font name. For instance, the font directory structure on the G-WINDOW release disk is as follows:

```
FONTS/
     Courier/       Helvetica/     Times/          fontlist

FONTS/Courier/
     Courier_11     Courier_13     Courier_15      Courier_18
     Courier_24

FONTS/Helvetica/
     Helvetica_9    Helvetica_11   Helvetica_13
     Helvetica_15   Helvetica_18   Helvetica_24

FONTS/Times/
     Times_11       Times_13       Times_15        Times_18
     Times_24
```

Note the file 'fontlist' in the 'FONTS' directory. It is a text file which contains the list of all fonts and their point sizes. this list is used by programs to quickly determine which fonts are available on the system. The

program '`fixfontlist`' is provided to maintain the list when new fonts are added. Simply run '`fixfontlist`' from any directory, and it will generate a new '`fontlist`' file based on the fonts stored currently on disk.

The following features are supported by G-WINDOW fonts:

1) mono or proportional spacing.
2) 1, 4, or 8 bits per pixel in font data (2, 16, or 256 colors per font).
3) automatic underline and bold attributes (currently only for G-WINDOWS 'quick fonts').
4) up to 256 by 256 pixels for each character in the font.
5) up to 256 characters in each font.

Although they could be implemented by an application program, the following features are NOT supported directly by G-WINDOW fonts:

1) Vertically aligned fonts.
2) Variable height for each character of the font (used with vertical fonts).
3) Fonts written right to left.
4) Rotating or displaying fonts at an angle.
5) Automatic italic attribute.

For a more detailed discussion of the actual construction of G-WINDOW fonts, please see the font section of the manual 'Developing Programs for G-WINDOWS'.

Chapter 6: Image Modules

In G-WINDOWS, 'images' refers to bit mapped non-repeating patterns and their associated color look-up table. Images are generally the output of paint programs or digitizers. G-WINDOW images may contain 1, 4, or 8 bits per pixel. They are stored as a rectangular picture, but some pixel data may be ignored when drawing to produce irregularly shaped images.

G-WINDOW images are stored in OS-9/68K data modules called 'image modules'. This is a format used only by G-WINDOWS. Rather than direct use of one of the graphics file formats supported by other operating systems, G-WINDOWS uses image modules for these reasons:

1) one copy of an image module may be shared by multiple processes, reducing memory requirements
2) an image module may be used directly from ROM - all other standards require an image to be loaded from a ROM disk into RAM, which uses more than twice the memory.
3) it is easy to create image modules, so importing files from another format is not difficult.

Image modules may contain multiple images. Each image within an image module is identified by a name and an image number. Image numbers are used when there are a series of images with the same name, as is the case when an animation sequence is stored. Each image within the module may specify its own CLUT, or may use a global CLUT which is specified at the beginning of the module.

Images may contain a maximum of 8 bits per pixel. Their CLUT may consist of up to 24 bits for each color entry. There is no size limit for the width and height of the image.

By convention, image modules are stored in the 'IMAGES' directory on the main disk device on the system. Programs which use image modules will

load modules from there first, and then will search the directories specified in the PATH environment variable.

Compress_Image

The program 'compress_image' is provided to compress or decompress G-WINDOWS image modules using a very fast run-length compression algorithm.

Compressed image modules may be used by all G-WINDOW programs and utilities just like a normal image module. Image decompression takes place automatically at run-time.

When a compressed image module is used, the memory which actually holds the image data is allocated by the user process instead of residing in the module itself. Because of this, a single compressed image which is in use by two different processes will take up twice the memory of the same image which was not compressed.

Compressed image modules gain advantage over uncompressed when many image modules are needed by an application, but only a few will be in actual use at any one time. Under these conditions, a lot of ROM or disk space may be saved at the expense of enough RAM to hold the needed images.

The calling format for 'compress_image' is as follows:

```
Syntax: compress_image [opts] [<filename>] [opts]
Options:
    -d                  Decompress image module
    -f=<pathlist>       Name of the output file
    -m=<name>           Name of the output image module
    -r                  Overwrite existing output file
```

Import_gif

The program 'import_gif' is provided to convert Compuserve GIF graphics files to G-WINDOW image modules. This provides G-WINDOWS with a very large, and growing, supply of public domain graphics files.

The calling format for 'import_gif' is as follows:

```
Syntax: import_gif [opts] [<GIF filename>] [opts]
Options:
    -c                  Trim CLUT size to fit greatest pixel
                        value
    -d                  Set image module to the smallest
                        pixel depth possible
    -d=<num>            Set image module pixel depth to <num>
                        (1, 4, or 8)
    -f=<pathlist>       Name of the output file
    -g=<num>            Translate image #<num> of the GIF
                        file (0-n)
    -i=<num>            Set output image ID number (1-65535)
    -m=<name>           Name of the output image module
    -n=<name>           Name of the image within the module
    -r                  Dynamically select area to save from
                        image
    -r=<lx,ly,ux,uy>    Save only this area of image
    -s                  Silent operation (no prompts or
                        messages)
```

The '-c' and '-d' options are used to store the image as efficiently as possible.

The '-r' option is used when only part of the image is to be save in the image module. If the coordinates are not included with the '-r' option, the image is displayed and the mouse is used to select the area to save (select the lower left corner followed by the upper right).

If a needed name is not supplied on the command line, '`import_gif`' will prompt the user at run time.

Savecrt

The program 'savecrt' is provided to convert parts of the screen display directly into image module format. With it, individual windows or the entire CRT may be saved as an image.

The calling format for 'savecrt' is as follows:

```
Syntax: savecrt [opts] [<window device>] [opts]
Options:
    -a              Save active window with frame
    -b              Save active window without frame
    -c              Save entire CRT (default)
    -f=<pathlist>   Name of output file
    -m=<name>       Name of output image module
    -n=<name>       Name of image within module
    -o              Do not optimize data
    -r              Overwrite output file
```

The output file option '-f' is a required parameter.

Normally, 'savecrt' attempts to optimize the data as it saves it so that the image module uses the least amount of space possible. This includes making all used CLUT entries consecutive starting with zero, and attempting to reduce the number of bits per pixel. If fast speed or the actual raw pixel values are required, use the '-o' flag.

As an example, this is one way to save the contents of the active window without saving the window frame as well:

```
savecrt /win -obf=FRED/imagefile
```

Viewimage

The program '`viewimage`' is provided as a simple means of viewing image modules. To view an image, simply run 'viewimage' with the name of each wanted image file as a command line argument. Up to 28 images may be viewed at once.

Images with a greater pixel depth than the graphics hardware supports cannot be viewed. For instance, a 256 color image cannot be viewed on a system that only supports 16 colors.

Chapter 7: Gadget Modules

G-WINDOW gadgets are self-contained and self-maintaining program pieces that are used to provide all manner of I/O devices in a window. There are gadget to provide each of the following and more: text entry fields, text buttons, image buttons, radio buttons, x boxes, text lists, state selectors, LEDs, bar graphs, scroll bars, and program variable output.

G-WINDOW gadgets are stored in OS-9/68K data modules called 'gadget modules'. This is a specific type of OS-9/68K subroutine module used only by G-WINDOWS. One copy of a gadget module may be shared by multiple processes, reducing memory requirements. Gadget modules may easily be used with ROMed systems.

If a program uses gadgets, the gadgets should be loaded into memory or be present in either your execution directory or the 'GADGETS' directory on your system's main disk drive.

If gadgets are not going to be used in your application program, they can be ignored for the most part. Programs which do use them will load them into memory and remove them when they are no longer needed.

The easiest way to incorporate gadgets into your programs is to use the G-VIEW development software. G-VIEW lets control windows which use gadgets be added to applications without requiring any graphics programming or in-depth knowledge of G-WINDOWS.

Appendix A: Escape Sequences for Normal Text

As part of the built in terminal emulation, the window manager recognizes a large subset of the VT100 escape sequence set, as well as some VT52 escape sequences. The following are the escape sequences the window manager recognizes when encountered through normal text output.

In all the following sequences, <ESC> represents the single ASCII character $1b. <SPACE> represents the single ASCII character $20. <CR> represents the single ASCII character $0d.

In all the following sequences, <Pn> should be replaced with the actual parameter value in a single byte. For instance, the value 123 (Hex $7b) would be sent to the window manager as the ASCII character '{'.

In all the following sequences, (Pn) should be replaced with the ASCII decimal version of the parameter value. For instance, the value 123 (Hex $7b) would be sent to the window manager as the three ASCII characters '1', '2', and '3' in sequence. Sometimes, a question mark (?) is required to appear in the escape sequence before the value.

The construct (P0);...;(Pn) indicates that any number of values may be specified in the escape sequence, separated by semi-colons.

The construct CTRL-<letter> should be replaced by the value obtained by subtracting $40 from the ASCII value of letter. This is the same value that is produced by the keyboard when the <CTRL> key is held while the <letter> key is pressed. For instance, CTRL-J is equivalent to the ASCII value $0a.

Communication With The Terminal

The following escape sequences may be used to query the state of terminal emulation. After the query escape sequence is sent to the window manager by your program, the response is returned as though it were entered at the keyboard.

Request Type	Query Sequence	Response Sequence
Terminal ID	<ESC><SPACE>	60<CR>
Terminal ID	<ESC>Z	<ESC>[?1;2c
Terminal ID	<ESC>[0c	<ESC>[?1;2c
Terminal Status	<ESC>[5n	<ESC>[0n
	Always responds with terminal ready message	
Cursor Position	<ESC>[6n	<ESC>[(P0);(P1)R
	(P0) = cursor row (1-n), (P1) = cursor column (1-n)	
Printer Status	<ESC>[15n	<ESC>[?11n
	Always responds with printer busy message.	

Controlling The Terminal

Terminal mode on `<ESC>[(P0);...;(Pn)h`

(Pn) = 3	enable monitor mode
(Pn) = 4	enable character insert mode (works only with text buffering turned on)
(Pn) = ?5	reverse screen on (will do strange things to dim or blinking characters already present on the screen)
(Pn) = ?7	enable end-of-line wrap
(Pn) = ?25	enable text cursor
(Pn) = 33	steady text cursor
(Pn) = 34	underline cursor

Terminal mode off `<ESC>[(P0);...;(Pn)l`

(Pn) = 3	disable monitor mode
(Pn) = 4	disable character insert mode
(Pn) = ?5	reverse screen off (will do strange things to dim or blinking characters already present on the screen)
(Pn) = ?7	disable end-of-line wrap
(Pn) = ?25	disable text cursor
(Pn) = 33	blinking text cursor
(Pn) = 34	block cursor

VT52 enable cursor	`<ESC>e`
VT52 disable cursor	`<ESC>f`
VT52 enable end-of-line wrap	`<ESC>v`
VT52 disable end-of-line wrap	`<ESC>w`
Reset all terminal modes	`<ESC>!p`
Reset terminal to initial state	`<ESC>c`
Sound bell and invert window briefly	`CTRL-G`

Save cursor position and attribute	<ESC>7
-or-	<ESC>[s
VT52 save cursor position	<ESC>j
Restore cursor pos. and attribute	<ESC>8
-or-	<ESC>[u
VT52 restore cursor to saved position	<ESC>k

Screen Display And Attributes

Define scrolling region `<ESC>[(P0);(P1)r`

`(P0)` = top row (1-n) of scrolling region

`(P1)` = bottom row (1-n) of scrolling region. If `(P1)` is zero or missing, the bottom row will be the last row on the screen.

Define video attribute `<ESC>[(P0);...;(Pn)m`

`(Pn)` =	0	all attributes off
`(Pn)` =	1	bold
`(Pn)` =	2	dim (not available with monochrome displays)
`(Pn)` =	4	underline
`(Pn)` =	5	blink (available only with CLUT displays)
`(Pn)` =	7	reverse
`(Pn)` =	8	Invisible
`(Pn)` =	22	all attributes off
`(Pn)` =	24	underline off
`(Pn)` =	25	blink off
`(Pn)` =	27	reverse off

VT52 reverse on `<ESC>p`

VT52 reverse off `<ESC>q`

Enable top half of double-high, double wide line `<ESC>#3`

Enable bottom half of double-high, double wide line `<ESC>#4`

Enable single-high, single-wide line `<ESC>#5`

Enable single-high, double-wide line `<ESC>#6`

Enable top half of double-high, `<ESC>#:`

single-wide line

Enable bottom half of double-high, single-wide line <ESC>#;

Controlling The Cursor

Cursor right one column	<ESC>[C
or VT52	<ESC>C
Cursor right (P0) columns	<ESC>[(P0)C
Cursor left one column	CTRL-H
Cursor left (P0) columns	<ESC>[(P0)D
Cursor up one line;no scroll (VT52)	<ESC>A
Cursor up one line;scroll	<ESC>M
or VT52	<ESC>I
Cursor up (P0) lines;no scroll	<ESC>[(P0)A
Cursor up (P0) lines and to column 1	<ESC>[(P0)F
Cursor down (P0) lines and to column 1	<ESC>[(P0)E
Cursor down one line;scroll	<ESC>D
- or -	CTRL-J
- or -	CTRL-K
- or -	CTRL-L
or VT52	<ESC>B
Cursor down (P0) lines; no scroll	<ESC>[(P0)B
Cursor to start of line (<CR>)	CTRL-M
Cursor to start of next line;scroll	<ESC>E
Cursor to column (P0) (1-n)	<ESC>[(P0)G

Cursor to row (P0); column (P1) range (1-n)	- or -	<ESC>[(P0);(P1)H <ESC>[(P0);(P1)f
VT52 cursor to row <P0>-$1f; column <P1>-$1f. range (1-n)		<ESC>Y<P0><P1>

Editing

Set tab stop at current position	<ESC>H
Clear tab stop at current position	<ESC>[0g
Clear all tab stops	<ESC>[3g
Move cursor to next tab stop, or right margin. This will only work when tmode is used to disable SCF style tabbing.	CTRL-I
Move cursor forward (P0) tab stops	<ESC>[(P0)I
Move cursor backward (P0) tab stops	<ESC>[(P0)Z
Insert (P0) null characters beginning at cursor column. Works only when window text buffering is enabled.	<ESC>[(P0)@
Insert (P0) lines of null characters beginning at cursor row	<ESC>[(P0)L
VT52 Insert 1 line of null characters	<ESC>L
Delete (P0) lines beginning at cursor row	<ESC>[(P0)M
Delete (P0) characters beginning at cursor column. Works only when window text buffering is enabled.	<ESC>[(P0)P
Erase from cursor to end of screen or VT52	<ESC>[0J <ESC>J
Erase from beginning of screen through cursor	<ESC>[1J

Erase entire screen;no cursor movement	<ESC>[2J
Erase entire screen and home cursor	CTRL-Z
Erase from cursor to end of line	<ESC>[0K
or VT52	<ESC>K
Erase from start of line through cursor	<ESC>[1K
or VT52	<ESC>o
VT52 clear from start of line to just before cursor	<ESC>d
Erase entire line	<ESC>[2K
or VT52	<ESC>l
Erase (P0) characters beginning at cursor column	<ESC>[(P0)X

Character Sets

Select G0 character set	CTRL-O
Select G1 character set	CTRL-N
Change G0 character set to ANSI Graphics set	<ESC>(0
Change G0 character set to UK ANSI set	<ESC>(A
Change G0 character set to Standard ANSI set	<ESC>(B
Change G1 character set to ANSI Graphics set	<ESC>)0
Change G1 character set to UK ANSI set	<ESC>)A
Change G1 character set to Standard ANSI set	<ESC>)B

Appendix B: Example Hibernate Icon File Source Code

The following is the partial assembly source code for a hibernate icon file to place in the '.ICONS' directory. The entire icon definition is not included here due to its size.

If the source file is named 'icon.a', and it is intended for use with the OS-9/68K shell, it should be assembled as:

```
r68 icon.a -o=/r0/icon.r
l68 -r /r0/icon.r -o=/h0/.ICONS/shell
attr -rpr /h0/.ICONS/shell
```

It is important to set the public read permission on the file.

The source code follows:

```
     use /h0/defs/oskdefs.d

Edition     equ 1
Typ_Lang    equ (Data<<8)+Objct
Attr_Rev    equ (ReEnt<<8)+0

     psect  substart_a,Typ_Lang,Attr_Rev,Edition,0,0

flags
* A long integer which includes:
*    Bit 0 = Put window title at the bottom of the
*                hibernate icon.
*    Bit 1 = We are including two icons.  The second will
*                be used when the program locks while
*                waiting for input.
*    Bit 2 = Right justify the window title within the
```

```
*                icon.
     dc.l   %0000000000000011

* two 64 by 64 pixel icons, each arranged as 4 columns of
*    64 words which are 16 pixels wide
busy_icon
     dc.w   %1111111111111111
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000000000
     dc.w   %1000000000011100
     dc.w   %1000000000100000
     dc.w   %1000000001000000
     dc.w   %1000000011000000
     dc.w   %1000000100000000
     dc.w   %1000001000000000
     dc.w   %1000011000000000
     dc.w   %1000100000000000
     dc.w   %1001000000000011
     dc.w   %1000100000000011
     dc.w   %1000010000000011
     dc.w   %1000001000000011
     dc.w   %1000001000000011
     dc.w   %1000000100000011
     dc.w   %1000000100000011
     dc.w   %1000000100000011
     dc.w   %1000001000000011
     dc.w   %1000001000000011
     dc.w   %1000000110000011
     dc.w   %1000000100000011
     dc.w   %1000000100000000
     dc.w   %1000000011000000
```

```
        dc.w    %1000000000111000
        dc.w    %1000000000000110
        dc.w    %1000000000000001
        dc.w    %1000000000000000
        dc.w    %1000000000000000
        dc.w    %1000000000000000
```

Part B: THE DESKTOP MANAGER

The desktop manager is a G-WINDOWS application program. If you are not familiar with the use and handling of windows, please read part A of this manual, "Using G-WINDOWS", before continuing.

The desktop manager is a program which replaces many of the file handling and program starting functions of the OS-9/68K shell. It is a visually oriented utility rather than a text oriented utility. While commands are given to the OS-9/68K shell with typed in command lines, most desktop manager commands are entered simply by pointing and clicking the mouse.

Disk devices are displayed as icons in the backdrop window. Opening a device (or directory residing on the device) causes a window to appear in which the files contained in the directory are displayed as small icons. Files may be moved, copied, deleted, edited, and executed without using the keyboard at all.

The desktop manager is not intended to replace the shell as a development tool, but to be used in conjunction with it. Since the desktop manager makes it very easy to perform the most common file functions, using OS-9/68K for the development of programs is greatly simplified.

Chapter 1: Setting Up the Desktop Manager

When started, the desktop manager looks for files in a directory named '.DESKTOP'. The '.DESKTOP' directory must be placed in the root directory of the device named in your system's init module. If no disk device is specified in the init module, the directory '/dd/.DESKTOP' is used instead. Make sure that the directory exists before running the desktop manager.

By using the '-d' command line option when starting the desktop manager, a different directory may be specified for use as the '.DESKTOP' directory. When more than one desktop manager is being used on the same computer, this option should be used to provide a private directory for each desktop manager.

The Bit Bucket Directory

A directory must be created for use as the bit bucket. Files are copied to the bit bucket when they are deleted. Since the bit bucket is treated almost like a normal disk device, this gives you a chance to recover files you have deleted. To free up disk space, the bit bucket directory can be emptied, at which times the files are actually deleted from the disk.

The bit bucket directory should be placed on the disk device that you use most often (usually '`/h0`'). When possible, put the bit bucket directory in the '`.DESKTOP`' directory.

The 'config' File

When the desktop manager starts up, it reads a file called 'config' in the '.DESKTOP' directory. The 'config' file describes what disk devices are used on your system, and also describes how different types of files will be executed. By editing the 'config' file, you can customize the way the desktop manager acts to your own tastes.

The 'config' file contains line-oriented text. Lines beginning with '#' and '*' are considered comments and are ignored. Don't get carried away putting in comments, though. The desktop manager doesn't remember them currently, and they will be lost when you use the 'save desktop' menu choice.

The Shell Line

The first line in the 'config' file defines what program will be used as the 'shell' program by the desktop manager. Usually this will specify the standard OS-9/68K shell program, but if you are used to using a non-standard shell, you may specify it here. The line format is as follows:

```
shell <shell program name>
```

where <shell program name> = the name of the program module to use instead of 'shell'. Note that this is not an entire pathlist, but just the program name itself.

The replacement shell will be used wherever a shell program is required by the desktop manager. The replacement shell must properly handle the following on the command line:

1) I/O redirection using <, >, +, and -
2) Setting priorities with ^
3) Forking programs concurrently with &
4) Parentheses
5) Quotes
6) Semi-colons

The Device Section

The device section of the 'config' file describes the disk devices present on your computer, including the bit bucket. Each line in this section describes a single device. The line format is as follows:

```
<device> <type> [<x position> <y position>]
```

where:

<device> = The name of the disk device or directory to be treated as a device. This field must begin with '/'. If the directory pathlist is not the root directory of a device, the '/' immediately following the root portion of the pathlist should be entered as '\' instead. This helps the desktop manager to decide when to move files instead of copying them.

<type> = The type of device. This field must be one of these four choices: hard, floppy, ram, and bucket.

<x position> and <y position> = The pixel location of the top center of the device icon. These fields may be omitted. The desktop manager will fill them in when the menu choice 'save desktop' is selected.

If there is more than one device descriptor that points to the same device, for instance '/h0' and '/dd', only one of them should be referenced in the 'config' file. Otherwise, the desktop manager may get confused.

The File Section

This section of the 'config' file describes how the different recognized types of files will be executed. The line format is as follows:

```
<file type> {[<custom recognizer>]} [<command line>]
```

where <file type> = One of the standard recognized file types. This field must begin with ':'. The defined file types are:

:text	unrecognized text file
:binary	unrecognized binary file
:module	unrecognized OS-9/68K module
:rof	relocatable object file
:prog	program module
:subr	subroutine module
:mult	OS-9/68K multi-module
:data	data module
:trap	trap handler module
:sys	system module
:fman	OS-9/68K file manager
:drvr	device driver
:desc	device descriptor
:make	makefile
:script	shell command file
:c	c source code
:asm	assembly source code
:font	G-WINDOW font module
:qfonts	G-WINDOW quick fonts module
:gadget	G-WINDOW gadget module
:recognizer	Desktop manager custom file recognizer
:image	G-WINDOW image module
:window	G-VIEW window module

<custom recognizer> = The name of a subroutine module which is used to extend the desktop manager so it can recognize more types of files.

This field must begin with ':'. After the ':', the full pathlist of the subroutine module must follow.

Up to eight subroutine modules may be attached to each basic file type. When a directory is opened, each file is checked to determine its basic type. Then each of the subroutine modules attached to that type will be called until one of them recognizes the file. The subroutine module then supplies its own icon and command line for executing the file.

<command line> = The shell command line which is used to execute the basic file type. If a file type is not to be executed at all, this field may be omitted.

When a file is executed, wherever '@' is encountered in the command line, the name of the file will be inserted. Likewise, whenever '%' is encountered, the name of the window descriptor the desktop manager uses (usually '/win') is inserted. These replacements are done even within quoted strings.

The standard I/O paths of the desktop manager are normally set to '/nil', so if you want the executed file to show up in a window, the I/O redirection must be included in the command line.

Sample 'config' File

```
shell shell
*
* Devices
*
/h0 hard
/d0 floppy
/r0 ram
/h0\.DESKTOP/BUCKET bucket
*
*
* File Types
*
:text    :/h0/wm/cmds/tcheck1      umacs @ <>>>%
:binary  :/h0/wm/cmds/bcheck
:module
:rof     rdump @ <>>>%
:prog    @ <>>>%
:supr
:mult
:data
:trap
:sys
:fman
:drvr
:desc
:make    make -f=@ ^100 <>>>%
:script  @ >>>%
:c       umacs @ <>>>%
:asm     umacs @ <>>>%
:font
:qfonts
:gadget
:recognizer
:image   viewimage @ <>>>%
:window  wedit @ <>>>%
```

The 'menu' File

When the desktop manager starts up, it reads a file called 'menu' in the '.DESKTOP' directory. If it exists, the 'menu' file describes additions to make to the menu available from the desktop manager. By editing the 'menu' file, the desktop menu can be customized to allow easy forking of commonly used programs.

Each line of the 'menu' file consists of two parts: the name to appear in the desktop menu, and a command line to run when that item is chosen. The command line is similar to that described in the 'config' file. Wherever '@' is encountered in the command line, the names of all currently selected files will be inserted. Likewise, whenever '%' is encountered, the name of the window descriptor the desktop manager uses (usually '/win') is inserted. These replacements are done even within quoted strings.

The standard I/O paths of the desktop manager are normally set to '/nil', so if you want the executed file to show up in a window, the I/O redirection must be included in the command line.

Sample 'menu' File

```
"Custom Choice #1"      dir -ea <>>>%
"Look at Text"          umacs -v @ <>>>%
"Park Disk Head"        hdpark /h0
"Fred Flintstone"       echo "@ Yabba Dabba Doo!!!!" <>>>%
```

The 'view' File

This file sets the default viewing style for directories that have not specifically had their style set with the desktop menu choice 'view'. It controls whether files will be displayed as text or icons, how they will be sorted, and whether files beginning with '.' will be shown. This file can be modified by using the 'view' menu choice when the backdrop window is active. This file can generally be ignored by the user.

The file '.dt_view' will appear in every directory that has had its viewing type set through the desktop menu. When the '.dt_view' file is present, its settings will override the 'view' file.

Adding Custom File Recognizers

Since it is impossible for the desktop manager to anticipate every type of file that will ever be used with it, a hook has been provided to customize its file recognition abilities. Through the use of OS-9/68K subroutine modules, the desktop manager can be expanded to provide icons and methods of execution for a large number of different file types.

This is useful when using programs like Dynacalc and Stylograph, which each create a unique kind of file when storing data to disk. When the desktop manager recognizes a file, it not only represents the file with an appropriate icon, but can execute the program that created it when the file icon is opened. For example, double clicking on a Dynacalc data file will cause Dynacalc to execute and load the file as data.

Up to eight separate subroutine modules may be attached to each standard file type defined in the config file. There is no limit to the number of custom file types each subroutine module may look for.

The subroutine module contains a C function which is called by the desktop manager whenever a file needs to be identified. The C function is written just like a normal C program except for the following exceptions:

1) Instead of the function '`main()`', the function executed is called '`dt_hook()`'. '`dt_hook()`' is passed a pointer to a MATCHTEST data structure (see '`desktop.h`').

2) No static storage may be used. Only '`register`' and '`auto`' variables declared WITHIN functions are allowed.

3) No I/O function calls may be made. This includes disk files, serial ports, pipes, and anything else.

4) Any functions which would change the state of the desktop manager are forbidden. This includes functions like '`exit()`', '`intercept()`', '`setuid()`', '`setpr()`', and '`chdir()`'.

5) The source code must be compiled with the '-ix' options of the 'cc' program. This makes it use the 'cio' and 'math' trap handlers at runtime. The '-r' option must also be used to stop compilation before the linking phase.

6) To create the actual subroutine module, the relocatable object file (the *.r file) must be linked with the 'substart.r' file in the following manner:

```
168 substart.r yourfile.r -o=yourfile -l=/dd/LIB/cio.l
    -l=/dd/lib/clibn.l -l=/dd/lib/sys.l
```

It is important to turn on both public and private execute permissions in the output file.

Using the file information passed to it in a MATCHTEST data structure, the 'dt_hook()' function must determine whether it can recognize the file as quickly as possible. The function may be called for every file in a large directory, so it is important to keep it as efficient as possible.

If no match is found, 'dt_hook()' simply returns to its caller. If the file is recognized, 'dt_hook()' must copy an icon pointer and a command line into the MATCHTEST data structure. It then sets the 'matchflag' structure element to 1 and returns.

See appendix B for a source code to a sample subroutine module and how to compile it.

Chapter 2: Using the Desktop Manager

The desktop manager is started from a command line in much the same way as the OS-9/68K utility 'tsmon'. It is started up as a background task with the name of the window descriptor supplied as a command line argument. The command line syntax for starting the desktop manager is as follows:

```
Syntax: desktop [opts] [window descriptor] [opts]
Options:
    -c              Use only two colors on CLUT systems
    -d=<pathlist>   Path to desktop configuration directory
    -i=<pathlist>   Name of image module to display in
                    backdrop
    -r              Reverse colors on monochrome backdrop
                    image
```

For example:

```
setenv PROMPT "@Yes? "
setenv _sh -1
setenv TERM WIN
setenv PATH /h0/GAMES/CMDS
desktop /win2 -i=gibson &
```

If the window descriptor is not included on the command line, the desktop manager uses the default descriptor '/win'.

By using the '-d' command line option when starting the desktop manager, a different directory may be specified for use as the '.DESKTOP' directory. When more than one desktop manager is being used on the same computer, this option should be used to provide a private directory for each desktop manager.

The '-i' and '-r' options will cause an image module to be displayed in the backdrop window. This is a useless feature, but it seems to make the desktop manager more fun to use.

The Status Window

The desktop manager opens a text window in the bottom left corner of the screen. This window is used to report on the status of several functions of the desktop. It may be hibernated into an icon if it is not wanted.

The status window is a named window which can be written to by other programs running on your computer.

Device Icons

The disk devices and the bit bucket defined in the 'config' file are displayed as icons on the screen.

The icons are shown highlighted when a device is 'selected'. Selected devices may be moved on the screen, copied to other devices, or opened to view the files contained in them.

Devices are selected with the same methods used to select files. Those methods will be described shortly.

By pressing and holding the left mouse button on a desk icon, it may be moved to a new position on the screen. The new positions may be saved to the 'config' file by using the 'save desktop' menu choice.

Opening Devices and File Directories

Disk and directory file icons which are selected may be opened either through the menu choice 'open' or by double clicking on the icon. Double clicking will work on icons which are not already selected.

If the device or directory is successfully opened, a window will appear in which the files contained in the directory are displayed as icons. Files with names beginning with a '.' are not displayed in the directory window normally.

Selecting Device and File Icons

Icons may be selected in the following three ways:

1) Click the left mouse button on the unselected icon. This will cause all other selected icons to be 'unselected'.

2) Press the space bar when the mouse pointer is on the icon. This will not unselect other icons. If the icon is already selected, this will deselect it.

3) Press and hold the left mouse button while moving the mouse to box the icons you want to select. Release the button when all wanted icons are within the box.

Opening and Running Files

File icons may be opened and executed either through the menu choice 'open' or by double clicking on the icon. Double clicking will work on icons which are not already selected.

Editing and Creating Text Files

Some text files do not cause a text editor to be run when they are executed, including makefiles and shell scripts. All text files may be edited by selecting the menu choice 'edit text' when the file icon is highlighted.

A new text file may be created by selecting the 'edit text' menu choice when no file icons are selected.

Creating a New File Directory

A new directory may be created in the currently active directory window by selecting the 'make directory' menu choice.

Moving and Copying Files

Files may be moved to another directory on the same device, or copied to another device by pressing the left mouse button on a selected icon and dragging it to its new location.

Normally, the desktop manager will not overwrite a file if you attempt to copy a file with the same name into its directory. When copying files into the bit bucket, however, it will always overwrite files with the same name.

Deleting Files

Files may be deleted by moving their icons into the bit bucket. If you do not overwrite the file or empty the bit bucket, the file may be retrieved by moving the icon back out of the bucket.

The bit bucket may be emptied by selecting the menu choice 'empty bit bucket' when the bucket is selected. After the bit bucket is emptied, the deleted files may not be recovered.

Duplicating Files

Files and directories may be duplicated by selecting their icons and making the menu choice 'duplicate'.

Forking a Shell

A shell may be started from the desktop manager at any time by making the menu choice 'fork shell'. If a directory window is currently active, that directory will be the shell's home directory. The HOME environment variable is automatically set for the shell.

Forking a Command Line

A shell command line may be started from the desktop manager at any time by making the menu choice 'fork process'. Any files or arguments currently selected in the desktop manager will be included as arguments at the end of the command line. If a directory window is currently active, that directory will be the command line's home directory.

Appendix A: Source Code for Example File Recognizer

This source code may be compiled and used as a file recognizer subroutine module with the desktop manager. It is designed to inspect text files, and declares a match when the first line begins with the string "fred".

The following command lines will compile and link it into a subroutine module, assuming the source file name is 'tcheck1.c'. The file 'substart.r' is supplied with the desktop manager.

```
cc -r -ix tcheck1.c
168 substart.r tcheck1.r -o=tcheck1 -l=/dd/LIB/cio.l
      l=/dd/lib/clibn.l -l=/dd/lib/sys.l
attr -epe tcheck1
```

The source code begins:

```
#include <desktop.h>


static int blackicon();   /* trick the compiler into
                             indexing off the program counter */

int dt_hook(mt)
register MATCHTEST *mt;
{
    register int i,j;

    do {
        /* ignore it if a match has already been flagged or it
           is not a text file.  */
        if (mt->matchflag || !(mt->flags&FT_TEXT))
            break;
```

```
        /* we need at least 5 chars including a carriage
           return */
        if (mt->filesize < 5)
           break;

        /* check for barney's buddy */
        if (strncmp(mt->data,"fred",4) &&
              strncmp(mt->data,"FRED",4))
           break;

        /* the file matched our specifications */

        /* set pointer to icon */
        mt->icon = (void *)blackicon;

        /* pass back our special command line */
        strcpy(mt->command,"list @ <>>>%");

        /* name the type of file it is (up to 15 letters) */
        strcpy(mt->filetype,"Fred Flintstone");

        /* tell the desktop manager to stop looking */
        mt->matchflag = 1;

    } while (0);

    /* return to the desktop manager */
}


#asm
* put the icon definitions after a C function to ensure that
it
*    is placed in a 'psect' by the C compiler.
*
blackicon
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
```

```
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000011111000000000000000
    dc.w    %0000000000000001111110000110000000000000
    dc.w    %0000000000111110000000000010000000000000
    dc.w    %0000000001000000000000000001000000000000
    dc.w    %0000000100000000000000000000100000000000
    dc.w    %0000001001001110110011101100100000000000
    dc.w    %0000010000100010010100100101000000000000
    dc.w    %0000010000110011001100110101000000000000
    dc.w    %0000010000100010010100100101000000000000
    dc.w    %0000001000100010010111011000100000000000
    dc.w    %0000000100000000000000000001000000000000
    dc.w    %0000000010000000000000000010000000000000
    dc.w    %0000000001110000000000001100000000000000
    dc.w    %0000000000001111111111110000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
whiteicon
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000000000000000000000000
    dc.w    %0000000000000000000011111000000000000000
```

```
    dc.w    %0000000000000111111111110000000
    dc.w    %0000000001111111111111110000000
    dc.w    %0000000011111111111111111000000
    dc.w    %0000000111000100110001001100000
    dc.w    %0000001111011010101101011100000
    dc.w    %0000001111001100110011010100000
    dc.w    %0000001111011010101101011100000
    dc.w    %0000000111011010100010011100000
    dc.w    %0000000011111111111111111000000
    dc.w    %0000000001111111111111110000000
    dc.w    %0000000000000111111111110000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000
    dc.w    %0000000000000000000000000000000

#endasm
```

Index